Deployment Patterns

WSO2 API Manager includes five main components as the Publisher, Store, Gateway, Traffic Manager and Key Manager. In a stand-alone APIM setup, these components are deployed in a single server. However, in a typical production setup, they need to be deployed in separate servers for better performance. Installing and configuring each or selected component/s in different servers is called a distributed setup.

This topic includes the following sections.

- Main components of a distributed setup
- WSO2 API Manager deployment patterns
- Scaling the Gateway

Main components of a distributed setup

The following diagram illustrates the main components of an API Manager distributed deployment.

![Diagram of API Manager components](image_url)

*Figure: Main components of an APIM distributed setup*

Let's take a look at each component in the above diagram. For more information on these components, see Architecture.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Gateway</td>
<td>This component is responsible for securing, protecting, managing, and scaling API calls. The API gateway is a simple API proxy that intercepts API requests and applies policies such as throttling and security checks. It is also instrumental in gathering API usage statistics. We use a set of handlers for security validation and throttling purposes in the API Gateway. Upon validation, it will pass Web service calls to the actual back end. If it is a token request call, it will then directly pass the call to the Key Manager Server to handle it.</td>
</tr>
<tr>
<td>API Store</td>
<td>This component provides a space for consumers to self-register, discover API functionality, subscribe to APIs, evaluate them, and interact with API publishers. Users can view existing APIs and create their own application by bundling multiple APIs together into one application.</td>
</tr>
<tr>
<td>API Publisher</td>
<td>This component enables API providers to easily publish their APIs, share documentation, provision API keys, and gather feedback on API features, quality, and usage. You can create new APIs by pointing to the actual back end service and also define rate-limiting policies available for the API.</td>
</tr>
</tbody>
</table>
API Key Manager Server
This component is responsible for all security and key-related operations. When an API call is sent to the Gateway, it calls the Key Manager server and verifies the validity of the token provided with the API call. If the Gateway gets a call requesting a fresh access token, it forwards the username, password, consumer key, and consumer secret key obtained when originally subscribing to the API to the Key Manager. All tokens used for validation are based on OAuth 2.0 protocol. All secure authorization of APIs is provided using the OAuth 2.0 standard for Key Management. The API Gateway supports API authentication with OAuth 2.0, and it enables IT organizations to enforce rate limits and throttling policies for APIs by consumer.

Login/Token API in the Gateway node should point to the token endpoint of Key Manager node. The token endpoint of the Key Manager node is at https://localhost:9443/oauth2endpoints/token. In a distributed setup, it should be https://<IP of key manager node>:/oauth2endpoints/token.

In a clustered environment, you use session affinity to ensure that requests from the same client always get routed to the same server.

Session affinity is not mandatory when configuring a Key Manager cluster with a load balancer. However, authentication via session ID fails when session affinity is disabled in the load balancer.

The Key Manager first tries to authenticate the request via the session ID. If it fails, the Key Manager tries to authenticate via basic authentication.

API Traffic Manager
The Traffic Manager helps users to regulate API traffic, make APIs and applications available to consumers at different service levels, and secure APIs against security attacks. The Traffic Manager features a dynamic throttling engine to process throttling policies in real-time, including rate limiting of API requests.

LB (load balancers)
The distributed deployment setup depicted above requires two load balancers. We set up the first load balancer, which is an instance of NGINX Plus, internally to manage the cluster. The second load balancer is set up externally to handle the requests sent to the clustered server nodes, and to provide failover and autoscaling. As the second load balancer, you can use an instance of NGINX Plus or any other third-party product.

RDBMS (shared databases)
The distributed deployment setup depicted above shares the following databases among the APIM components set up in separate server nodes.

- **User Manager Database**: Stores information related to users and user roles. This information is shared among the Key Manager Server, Store, and Publisher. Users can access the Publisher for API creation and the Store for consuming the APIs.
- **API Manager Database**: Stores information related to the APIs along with the API subscription details. The Key Manager Server uses this database to store user access tokens required for verification of API calls.
- **Registry Database**: Shares information between the Publisher and Store. When an API is published through the Publisher, it is made available in the Store via the sharing registry database.

Message flows
The three main use cases of API Manager are API publishing, subscribing and invoking. Described below is how the message flow happens in these use cases.

- **Publishing APIs**
  A user assigned to the publisher role can publish APIs. This is done via the Publisher server. When an API is published in the API Publisher, it will be available in the API Store. Furthermore, the API Gateway must be updated with this API so that users can invoke it. As we are using a clustered Gateway, all Gateway server nodes in the cluster are updated with the published API details, enabling any of these Gateway nodes to serve API calls that are received. When an API is published, it is also pushed to the registry database, so that it can be made available on the store via the shared database.

- **Subscribing to APIs**
  A user with the subscriber role logs into the API Store and subscribes to an API. The user must then generate an access token to be able to invoke the API. When the subscriber requests to generate the token, a request is sent to the Key Manager Server cluster. The token is then generated, and the access token details are displayed to the subscriber via the Store.

- **Invoking APIs**
  Subscribed users can invoke an API to which they have subscribed. When the API is invoked, the request is sent to the Gateway server cluster. The Gateway server then forwards the request to the Key Manager server cluster for verification. Once the request is verified, the Gateway connects to the back-end implementation and obtains the response, which is sent back to the subscriber via the Gateway server.

The diagram below summarizes these message flows where the Publisher is referred to as the publishing portal and the Store is referred to as the developer portal.
Figure: WSO2 API Manager message flows

WSO2 API Manager deployment patterns

Pattern 1

Single node (all-in-one) deployment.
You can use this pattern when you are working with a low throughput.

**Pattern 2**
Deployment with a separate Gateway and separate Key Manager.

You can use this pattern when you require a high throughput scenario that requires a shorter token lifespan.

**Pattern 3**
Fully distributed setup.
You can use this pattern to maintain scalability at each layer and higher flexibility at each component.

**Pattern 4**

Internal and external (on-premise) API Management.
You can use this pattern when you require a separate internal and external API Management with separated Gateway instances.

**Pattern 5**

Internal and external (public and private cloud) API Management.

You can use this pattern when you wish to maintain a cloud deployment as an external API Gateway layer.

**Traffic Manager scalable deployment patterns**

See the article on Scalable Traffic Manager deployment patterns part 1 and part 2.

**Scaling the Gateway**

Scaling the Gateway requires considering the load that the Gateways will handle. We recommend that a load test be carried out on the Gateways in the environment. For more information on what factors affect the load, and how the numbers have to be derived in order to scale, refer to the article on Capacity Planning. Make sure to check the backends as well for the expected load.

**Clustering Gateways and Key Managers with key caching**

For key validation, the Gateway can usually handle 3,000 transactions per second (TPS), whereas the Key Manager can only handle 500 TPS. To improve performance, the key cache is enabled on the Gateway by default, which allows the system to handle 3,000 TPS. However, if you need better security, you can enable the cache on the Key Manager instead. Note the following about clustering with key caching:

- When the cache is enabled at the Gateway, you can have two Gateways per Key Manager.
- When the cache is enabled at the Key Manager and disabled at the Gateway, you can have only one Gateway per Key Manager.
- If both caches are disabled (not recommended), even with only one Gateway per Key Manager, the system may not be able to handle the load, as the Key Manager will only be able to handle 500 TPS.

For more information, see Key cache in the WSO2 API Manager documentation.
Scaling Gateways and Traffic Managers

Based on the performance numbers attained at the WSO2 Labs, the recommended ratio is 10 Gateways to 1 Traffic Manager (10:1, one Traffic Manager is tested to have successfully handled the load of 10 Gateways, one Gateway handling the load of ~5000 TPS). Note that Traffic Managers do not scale horizontally. If the requirement arises to have more than 10 Gateways, WSO2 recommends to have Traffic Managers placed in dedicated clusters of a maximum of 10 Gateways each. However, the number of Gateways in a cluster is not a hard limit and can vary according to your requirement.